# Introduction to Big data, Hadoop, hive and spark

# Jialiang Jiang

Big data itself is a very broad concept, and the Hadoop ecosystem is basically created to handle data processing beyond what one machine can do. You can compare it to a variety of tools that are needed in a kitchen. Pots and pans, each with its own use, overlap each other. You can use the stockpot to eat soup directly in the bowl. You can peel it with a knife or a plane. But each tool has its own characteristics, and although strange combinations work, they are not necessarily the best choice.

Big data, first you must be able to save big data.

Firstly, the traditional file system is offline and cannot span different machines. HDFS (Hadoop Distributed FileSystem) is designed to span a large number of data across hundreds of machines, but what you see is a file system rather than a lot of file systems. For example, if you say that I want to get the data of “/hdfs/tmp/file1”, you are referring to a file path, but the actual data is stored on many different machines. As a user, you don't need to know this. It's like you don't care about what sectors of the track are scattered on a single machine. HDFS manages this data for you.

After saving the data, you’ll need to start thinking about how to process the data.

Although HDFS can manage the data on different machines for you as a integrity, the data is too large. A single machine need to read the data in size of PB on TB. It may take several days or even weeks for a machine to run. For many companies, single thread is unbearable. For example, if Facebook wants to update the 24-hour recommendation blogs, it must finish the processing within 24 hours. Therefore, we need to figure out how to assign works to each machine. Similar questions like If one machine shut down, how to restart the corresponding tasks; how does the machines communicate with each other to exchange data to complete complex calculations and so on. This is the function of MapReduce, Tez and Spark. MapReduce is the first generation of computing engines, and Tez and Spark are the second generation. MapReduce's design uses a very simplified computational model. Only Map and Reduce are two computational processes (shuffled in the middle with Shuffle). With this model, a large part of the big data domain can be handled.

So, what is Map and what is Reduce?

Consider if you want to know how often each word appears in a huge text file stored on something like HDFS, you’ll need to use MapReduce. In the Map phase, hundreds of machines simultaneously read the various parts of the file, and separately counted the respective words to read the word frequency, resulting in form like Pair (hello, 12100 times), (world, 15214 times). Each of these hundreds of machines produces outcome, and then hundreds of machines start to do the Reduce. Reducer machine A will receive all the results starting with “A” from the Mapper machine, and machine B will receive all the results starting with “B” (of course, it will not actually start with the letter, but use a function to generate the hash value, because words like the beginning of “X” are definitely much less than others, and you don't want the amount of work assigned for each machine varies a lot). Then these Reducers will be aggregated again, (hello, 12100) + (hello, 12311) + (hello, 345881) = (hello, 370292). Each Reducer is processed as above, and you get the word frequency results for the entire file.

This seems to be a very simple model, but many algorithms can be described with this model.

The model of Map + Reduce seems very straightforward and simple, although it is easy to use, but it is also cumbersome. The second generation are Tez and Spark, in addition to the new feature such as memory Cache, essentially, it makes the Map/Reduce model more versatile, making the boundaries between Map and Reduce more ambiguous, the process of data exchange more flexible, and less disk reads. Write to make it easier to describe complex algorithms and achieve higher throughput.

With MapReduce, Tez and Spark, programmers found that MapReduce programs are still hard to write. They want to simplify this process. It's like you have basic natural language, although you can do almost anything, but you still feel troublesome. You want a higher level and more abstract language layer to describe the algorithm and data processing flow. Then there is Pig and Hive. Pig is close to scripting to describe MapReduce, and Hive is using SQL. They translate scripts and SQL language into MapReduce programs, throw them to the calculation engine for calculation, and you get rid of the cumbersome MapReduce programs and write programs in a simpler and more intuitive language.

With Hive, people found that SQL has a huge advantage over Java.

It’s literally easy to write. The word frequency question we’ve mentioned before, it’ll be only one or two lines in SQL description whether MapReduce writes about tens of hundreds of lines. More importantly, users who are not computer backgrounds finally feel love: I can also write SQL! Hive has grown into a core component of big data warehouses. Even many of the company's assembly line set is completely described in SQL, because it is easy to write and easy to change, easy to understand by just one glance.

Since data analysts began analyzing data with Hive, they found that Hive ran on MapReduce, which was slow! The pipelined job set may not be relevant, such as a 24-hour update recommendation blogs, sooner or later, it will have the result within 24 hours. But for data analysis, people always want to run faster.

There, we have Impala, Presto, Drill.

The core idea of ​​the three systems is that the MapReduce engine is too slow, because it is too versatile, too strong, and too conservative. Our SQL needs to be lighter, more aggressively accessing resources, and more specifically optimizing SQL without too much demands on fault tolerance guarantee. These systems allow users to process SQL tasks more quickly, at the expense of general stability. If MapReduce is a machete, it is not afraid to cut it. The top three are boning knives, smart and sharp, but you can't do anything too hard.

These systems, to be honest, have not reached the prevailing popularity. Because at this time two other unique were created. They are Hive on Tez / Spark and SparkSQL. Their design philosophy is that MapReduce is indeed slow, but if I run SQL with the next-generation general-purpose computing engine Tez or Spark, then I can run faster. And the user does not need to maintain two systems. This is like if your kitchen is small and you are lazy, but you are demanding on food. Then you can buy a rice cooker, it can steam and burn, and save a lot of kitchen utensils.

The above introduction is basically a framework of a data warehouse. The underlying layer is HDFS; MapReduce/Tez/Spark are above; Hive, Pig are higher. Or running Impala, Drill, and Presto directly on HDFS. This solves the requirements of low and medium speed of data processing.

Then what if I want to handle it at a higher speed?

Suppose we are doing a Facebook-like company and I want to maintain a table where users can see a constantly changing highly ranked blog list. The update delay is within one minute and the above methods will not be sufficient. Then another computational model was developed, which is the Streaming calculation. Storm is the most popular streaming computing platform. The idea of ​​stream computing is that if you want to achieve more real-time updates, why not deal with the data flow when it comes in? Let’s do the example of word frequency statistics, the data stream is a word, it let them flow through and start counting. The flow calculation is very powerful, basically no delay, but its shortcoming is that it is not flexible. The things you want to count must be known in advance. After all, the data flow will be gone once it flew through, and you can't make up for anything you haven't counted. So it's a good thing, but it's not a replacement for the data warehouse and batch system above.

In addition, there are some more special systems/components, such as Mahout, a distributed machine learning library, Protobuf, a data exchange encoding and library, ZooKeeper, a highly consistent distributed access collaboration system, and so on.

With so many messy tools all running on the same cluster, everyone needs to respect each other in an orderly manner. Thus, another important component is the scheduling system. The most popular is Yarn. You can think of it as a central management, like your mother is supervising the kitchen ---- “Hey, after your sister cut the vegetables and you can put them into the bowl. As long as everyone obeys your mother's orders, everyone can cook the dishes in order without making mess.

From all we have talked before, you can think that the big data ecosystem is a kitchen tool ecosystem. To make different dishes like Chinese, Japanese or French, you need a variety of different tools. And the needs of the guests are complicated, your kitchen utensils are constantly being invented, and no universal kitchen utensils can handle all situations, so it will become more and more complicated.

Look before you jump.